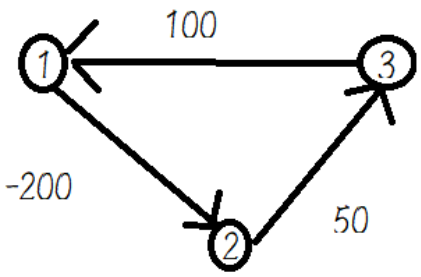
题目描述

7.18 请给出一个有向图的例子，使它包含一些负耗费的边，而所有点对最短路径算法不能给出正确的距离。

解答过程



存在负权边回路的有向图

带有“负权回路”的图没有最短路。例如下面这个图就不存在1号顶点到3号顶点的最短路径。因为1->2->3->1->2->3->...->1->2->3这样路径中，每绕一次1->-2>3这样的环，最短路就会减少1，永远找不到最短路。其实如果一个图中带有“负权回路”那么这个图则没有最短路。

题目描述

7.30 考虑金钱兑换问题。有一个货币系统，它有 n 种硬币，它们的面值为 $v_1, v_2, \cdots, v_n$，其中 $v_1 = 1$。我们想这样来兑换价值为 y 的钱，要让硬币的数目最少。更形式地，我们要让下面的量

$$\sum_{i=1}^{n} x_i$$

在约束条件

$$\sum_{i=1}^{n} x_i v_i = y$$

下极小。其中，$x_1, x_2, \cdots, x_n$ 是非负整数（x_i 可能是 0）。

(a) 设计求解这个问题的动态规划算法；

(b) 你的算法的时间和空间复杂性是什么；

(c) 你知道这个问题和练习 7.27 中讨论的背包问题的相似之处吗？请解释。

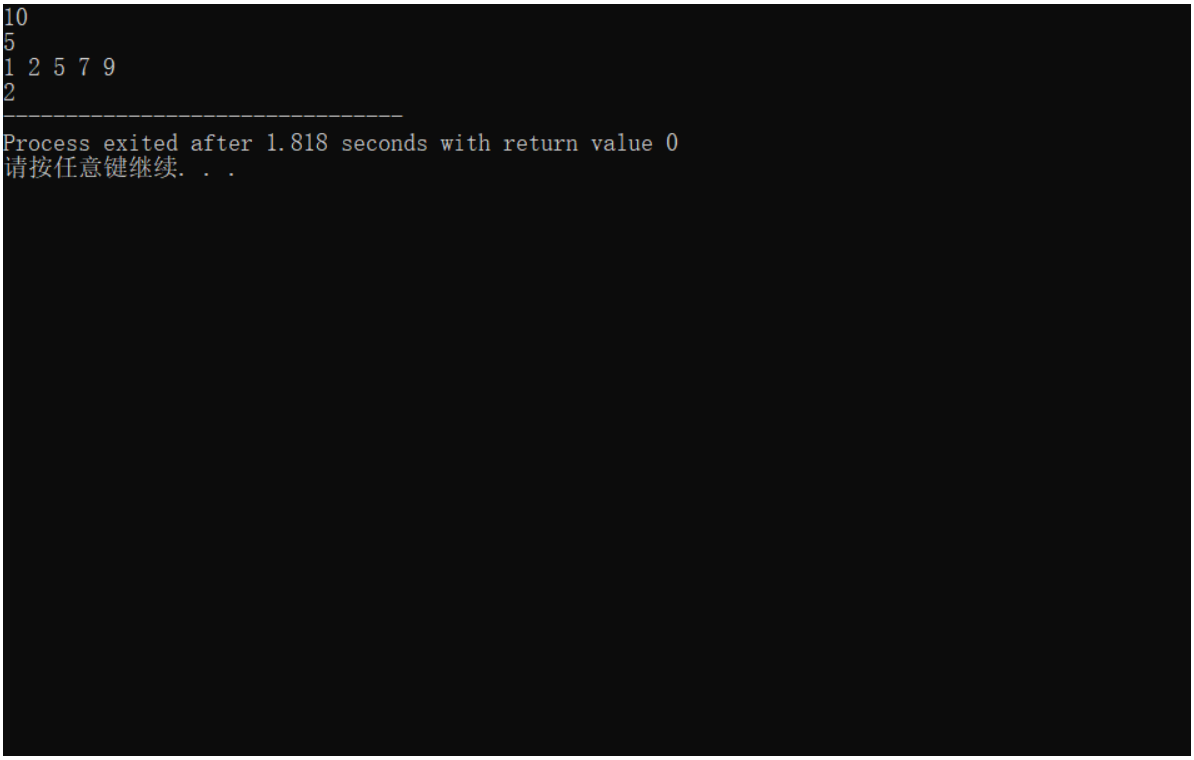
一维数组优化01背包问题

1. 设$c[i],j=0...y$示用该硬币系统找钱j的最少硬币个数。对于任何 0⩽j⩽y 及0⩽i⩽n，若j-vi>0,则$c[j-v[i]]+1$所表示的找钱$j-v[i]$的最优序列，再加1枚面值位$v[i]$的硬币是一种找钱j的方法，且所用的硬币个数为$c[j-v[i]]+1$。由此可得以下的递归式:$c[j]=min(c[j],c[j-v[i]]+1)$

   算法实现如下:

```
/*
10
5
1 2 5 7 9
*/
#include<iostream>
using namespace std;
const int N=110;
int f[N];
int v[N];
int n;
int MIN(int num1,int num2)
{
    if(num1!=-1&&num2!=-1)
    {
        if(num1<num2)
        return num1;
        else return num2;
    }
    else {
        if(num2==-1)
        return num1;
        return num2;
    }
}
int main()
{
    int y;
    cin>>y;
    cin>>n;
    for(int i=1;i<=n;i++) cin>>v[i];
    for(int j=1;j<=y;j++) f[j]=-1;
    for(int i=1;i<=n;i++)
    {
        for(int j=v[i];j<=y;j++)
        {
            f[j]=MIN(f[j],f[j-v[i]]+1);
        }
    }
    cout<<f[y];
    return 0;
}
```

运行截图:



2. 算法时间复杂度为 $\theta(ny)$,空间复杂度为$O(y)$
3. y相当于背包容量，n种硬币相当于n种体积为$v[i]$、价值为1的物品。但该题要求背包恰好装满，且总价值最小。

题目描述

7.32　设 $G=(V,E)$是一个有 n 个顶点的有向图，G 在顶点集合 V 上导出一个关系 R，它是这样定义的：$u\,R\,v$ 当且仅当从 u 到 v 存在一条有向边，即当且仅当$(u,v)\in E$。设 M_R 是 G 的邻接矩阵，即 M_R 是一个 $n\times n$ 矩阵，如果$(u,v)\in E$，则 $M_R[u,v]=1$，否则为 0。M_R 的自反和传递闭包，记为 M_R^*，定义如下，对于 $u,v\in V$，如果 $u=v$ 或 G 中存在一条从 u 到 v 的路径，那么 $M_R^*[u,v]=1$，否则为 0。对于给定的有向图，请设计一个动态规划算法来计算 M_R^*（提示：仅需对 Floyd 计算所有点对的最短路径算法稍做修改）。

算法实现

```
map[N][N];
memset(map,0,sizeof map)
void Floyd()
{
    //传递闭包算法
    for(int k=1; k<=n; k++)
        for(int i=1; i<=n; i++)
            if(map[i][k]==1)
                for(int j=1; j<=n; j++)
                    if(map[k][j]==1)
                        map[i][j] =1;

    //求自反闭包的算法
```

```
    for(int i=1;i<=n;i++)
    {
            map[i][i]=1;
    }
}
```